MC48 Lab 3:
Measuring Instruction Set Usage

Max Hailperin

Due: November 16, 1995

In this lab, you will work together with a partner to gain experience in how empirical measurements can be made of how real programs make use of an instruction set architecture, and how this data can then be used to evaluate the likely impact of proposed changes.

Because there is an odd number of students, exactly one group should be of one or three rather than of two. Please try to find your own partners; if you want to be the group of one or three (with two others), or if you can't find a partner, see me. Each group should submit a single jointly authored lab report.

This lab needs to be done on one of the SGI computers, not a NeXT, because it relies on using the real live MIPS processor. More precisely, the experimental measurements need to be done on an SGI; the majority of the lab will actually consist of analysis and writeup, which you can do on any machine.

You will use two programs that we have on the SGIs to do your measurements: `pixie` and `pixstats`. The `pixie` program reads a machine language program (i.e., a ready-to-run, executable program) and writes a variant of it that does all the same things but also keeps track of statistical information, such as how often each section of code is executed. For example, if you have a program called `gtroff` in your current directory and and you run `pixie gtroff`, then you will find that you now have a `gtroff.pixie` program in the current directory as well, which is the "pixified" version of `gtroff`. [There will also be one or more pixified libraries in your current directory, with names starting with `lib` and ending with `.pixie`; more on those later.] If you now run the `gtroff.pixie` program, it will do all the things that `gtroff` would, but will also leave a `gtroff.Counts` file in your current directory, which contains the raw statistical data from the run.

If you now do `pixstats -op gtroff`, the `pixstats` program will process that raw data and output a huge volume of information about the run. You probably should put that in a file, e.g. `gtroff.stats`, by doing

```
pixstats -op gtroff >gtroff.stats
```

You will be using `pixie` and `pixstats` in this way to measure two different real programs doing the same job, namely formatting some text. (Formatting means doing such things as breaking the paragraphs into lines and the document into pages.) You will measure the `gtroff` and `tex` programs, which are two popular publicly available text formatters. You will make your measurements while each of these formatting programs formats the exact same sample text, namely `~max/MC48/sample.text`. In both cases you will be measuring the version of the formatting program that is currently installed on our SGIs for real use. (Of course, you'll be making a pixified version, but the measurements will reflect the installed version.)

On an SGI, open up a shell window for doing the measurement work in, create a new directory for the purpose, and change directory into that directory. (Let me know if you need help with this kind of thing.)

Now, you want to have two existing files in this directory, namely the normal, un-pixified `tex` and `gtroff` programs. However, it would be wasteful to actually make your own copies. Therefore, what you should do is just put "symbolic links" into your directory, which make the files appear to be there but don't actually copy them. You would do this as follows:

```
ln -s /usr/local/bin/tex /usr/local/bin/gtroff .
```

At this point, you can run `pixie` on each of `tex` and then `gtroff`, as indicated above. You will find that you not only have the `tex.pixie` and `gtroff.pixie`, but also lots of pixified libraries. These libraries contain all sorts of common shared code, like the conversion of an integer into its constituent digit characters. To be fair, these libraries need to be included in the measurement, because what one program does "by hand" the other might do with a library routine. At any rate, to arrange that when you run the pixified programs they can load the pixified libraries they need, you should do the following command:

```
setenv LD_LIBRARY_PATH .
```

Now you are ready to run the two pixified text formatters on the sample text, to generate the two `.Counts` files with the raw statistics. (You'll

also generate two formatted versions of the text, for what it's worth.) For `gtroff.pixie`, the only magic is that you have to put the formatted version into a file using the shell's `>` output redirection operator. For `tex.pixie`, even more magic is needed, as shown below. In full, here are the two commands:

```
gtroff.pixie ~max/MC48/sample.text >sample.gt
tex.pixie '&tex' ~max/MC48/sample.text '\end'
```

These will leave you with the two statistical files, `gtroff.Counts` and `tex.Counts`. You'll also have the two formatted versions in `sample.gt` and `sample.dvi`; if you want to view those formatted versions and need help doing so, let me know.

Now you need to run `pixstats` twice as shown at the beginning of the lab handout to generate the two human-readable statistical summaries from the raw `.Counts` files.

Once you have done this, you should delete everything but the human-readable statistical summaries, since the pixified versions take up a fair bit of disk space. You should also exit out of the shell you've been using, because the `setenv` command you did earlier could cause problems if you continue to use this shell.

Now, analyze the statistical data you got in order to answer the questions listed below. You may well need help understanding the output; please ask. Write your answers to the following questions in the form of a readable report that explains the basis for your answers in terms of the data; if you had to make some approximations or guesses, make those explicit.

1. Suppose the MIPS architecture were extended to support the `blt` and `bge` operations in hardware. How much could each of the two programs benefit from this?

2. Suppose the MIPS architecture were simplified so that the only shift instructions shifted left or right by one bit position. What would the performance impact on each program be?

3. Some architectures otherwise similar to the MIPS have "register windows" that eliminate most of the need for saving registers to the stack on procedure entry and restoring them on procedure exit. (Instead, procedure calls simply select a new set of registers and procedure returns go back to using the old registers.) Of course, register windows have limitations and there is some performance penalty associated with

them. None the less, it is interesting to get an approximate bound on how much performance improvement would be possible from perfect free register windows. How much does each program stand to gain?

4. Instead of changing the architecture, it is possible to identify the most performance critical procedures and hand-optimize the code in those procedures. Suppose you could afford the time to do this for the 10 most performance-critical procedures, and managed to shave 10% of their time off. What would the overall performance improvement be for each program?

5. Suppose that the MIPS architecture were changed so that branch displacements were measured in bytes rather than in words, i.e., so that the shift left by 2 was eliminated from the datapath. This would effectively reduce the distance reachable with a branch instruction, which might require turning some long-distance branches into short branches around jumps. Suppose this architectural simplification was done; what would the performance impact on each program be?

6. Are there any other major differences between the two programs' behaviors that stand out to you from the statistical information you have, beyond those that you addressed in the prior questions?